

Quilting Stochastic Kronecker Product Graphs to Generate Multiplicative Attribute Graphs

Hyokun Yun

Department of Statistics
Purdue University

S.V.N. Vishwanathan

Department of Statistics and Computer Science
Purdue University

Abstract

We describe the first sub-quadratic sampling algorithm for the Multiplicative Attribute Graph Model (MAGM) of [Kim and Leskovec \(2010\)](#). We exploit the close connection between MAGM and the Kronecker Product Graph Model (KPGM) of [Leskovec et al. \(2010\)](#), and show that to sample a graph from a MAGM it suffices to sample small number of KPGM graphs and *quilt* them together. Under a restricted set of technical conditions our algorithm runs in $O((\log_2(n))^3 |E|)$ time, where n is the number of nodes and $|E|$ is the number of edges in the sampled graph. We demonstrate the scalability of our algorithm via extensive empirical evaluation; we can sample a MAGM graph with 8 million nodes and 20 billion edges in under 6 hours.

1 Introduction

In this paper we are concerned with statistical models on graphs. Typically one is interested in two aspects of graph models, namely scalable inference and efficiency in generating samples. While scalable inference is very important, an efficient sampling algorithm is also critical:

- To assess the goodness of fit, one generates graphs from the model and compares graph statistics of the samples with the original graph ([Hunter et al. 2008](#)).
- To test whether a certain motif is overrepresented in the graph, one strategy is to sample large number of graphs in the null hypothesis to approximate the p -value of the test statistic ([Shen-Orr et al. 2002](#)).

- To predict the future growth of the graph, one may fit the model on the current graph and generate a larger graph with the estimated parameters.

The stochastic Kronecker Product Graph Model (KPGM) was introduced by [Leskovec et al. \(2010\)](#) as a scalable statistical model on graphs. Compared to previous models such as Exponential Random Graph Models (ERGMs) ([Robins et al. 2007](#)) or latent factor models ([Hoff 2009](#)), KPGM sports a number of advantages. In particular, the inference algorithm of KPGM is scalable to very large graphs, and sampling a graph from the model takes time that is proportional to the expected number of edges. This makes the KPGM a very attractive model to study. However, [Moreno and Neville \(2009\)](#) report that KPGM fails to capture some characteristics of real-world graphs, such as power-law degree distribution and local clustering.

In order to address some of the above shortcomings and to generalize the expressiveness of KPGM, [Kim and Leskovec \(2010\)](#) recently proposed the Multiplicative Attribute Graph Model (MAGM). MAGM can provably model the power-law degree distribution while KPGM can not. Furthermore, [Kim and Leskovec \(2010\)](#) demonstrate empirically that MAGM is better able to capture graph statistics of real-world graphs ([Kim and Leskovec 2011](#)). The issue of inference for MAGM is addressed by [Kim and Leskovec \(2011\)](#) via an efficient variational EM algorithm.

However, the issue of *efficiently* sampling graphs from MAGM remains open. Currently, all algorithms that we are aware of scale as $O(n^2)$ in the worst case, where n is the number of nodes. This is because, the probability of observing an edge between two nodes is determined by the so-called $n \times n$ edge probability matrix. Unlike the case of KPGM where the edge probability matrix has a Kronecker structure which can be exploited to efficiently sample graphs in expected $O(\log_2(n)|E|)$ time, where $|E|$ is the number of edges, no such results are known for MAGM. In the absence of any structure, naively sampling each entry of the adjacency matrix requires $O(n^2)$ Bernoulli trials, which is prohibitively expensive for gener-

ating real-world graphs with millions of nodes.

In this paper we show that under a restricted set of technical conditions, with high probability, a significant portion of the edge probability matrix of MAGMs is the same as that of KPGMs (modulo permutations). We then exploit this observation to *quilt* $O((\log_2(n))^2)$ graphs sampled from a KPGM to form a single sample from a MAGM. The expected time complexity of our sampling scheme is thus $O((\log_2(n))^3|E|)$.

1.1 Notation and Preliminaries

A graph G consists of an ordered set of n nodes $V = \{1, 2, \dots, n\}$, and a set of directed edges $E \subset V \times V$. A node i is said to be a neighbor of another node j if they are connected by an edge, that is, if $(i, j) \in E$. Furthermore, for each edge (i, j) , i is called the source node of the edge, and j the target node. We define the adjacency matrix of graph G as the $n \times n$ matrix A with $A_{ij} = 1$ if $(i, j) \in E$, and 0 otherwise. Also, the following standard definition of Kronecker product is used (Bernstein 2005):

Definition 1 Given real matrices $X \in \mathbb{R}^{n \times m}$ and $Y \in \mathbb{R}^{p \times q}$, the Kronecker product $X \otimes Y \in \mathbb{R}^{np \times mq}$ is

$$X \otimes Y := \begin{bmatrix} X_{11}Y & X_{12}Y & \dots & X_{1m}Y \\ \vdots & \vdots & \vdots & \vdots \\ X_{n1}Y & X_{n2}Y & \dots & X_{nm}Y \end{bmatrix}.$$

The k -th Kronecker power $X^{[k]}$ is $\otimes_{i=1}^k X$.

2 Kronecker Product Graph Model

The Stochastic Kronecker Product Graph Model of Leskovec et al. (2010) is usually parametrized by a 2×2 initiator matrix

$$\Theta := \begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix}, \quad (1)$$

with each $\theta_{ij} \in [0, 1]$, and a size parameter $d \in \mathbb{Z}^+$. For simplicity, let the number of nodes n be 2^d . The case where $n < 2^d$ can be taken care by discarding $(2^d - n)$ number of nodes later as discussed in Leskovec et al. (2010), but in our context we will only use $n = 2^d$ for KPGM.

The $n \times n$ edge probability matrix P is defined as the d -th Kronecker power of the parameter Θ , that is,

$$P = \Theta^{[d]} = \underbrace{\Theta \otimes \Theta \otimes \dots \otimes \Theta}_{d \text{ times}}. \quad (2)$$

The probability of observing an edge between node i and j is simply the (i, j) -th entry of P , henceforth denoted as P_{ij} . See Figure 1 (left) for an example of the edge probability matrix of a KPGM, and observe its fractal structure which follows from the definition of P .

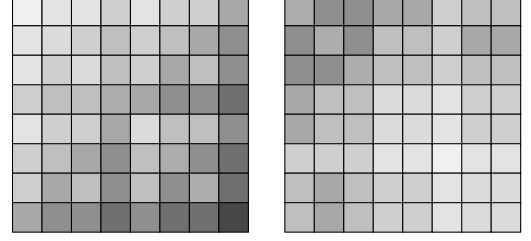


Figure 1: Examples of edge probability matrix (Left: KPGM, Right: MAGM). Darker cells imply a higher probability of observing an edge. On the left, one can see the fractal-like Kronecker structure; dividing the matrix into four equal parts yields four sub-matrices which up to a scaling look exactly the same.

Note that one can generalize the model in two ways (Leskovec et al. 2010). First, one can use larger initiator matrices. Second, different initiator matrices $\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(d)}$ can be used at each level. In this case, the edge probability matrix becomes

$$P = \Theta^{(1)} \otimes \Theta^{(2)} \otimes \dots \otimes \Theta^{(d)}. \quad (3)$$

In this paper we will work with (3) because it is closer in spirit to the MAGM which we will introduce later. For notational simplicity, we will denote

$$\tilde{\Theta} := \{\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(d)}\}. \quad (4)$$

2.1 Sampling

The straightforward way to sample a graph from a KPGM is to *individually* sample each entry A_{ij} of the adjacency matrix A independently. This is because, given the parameter matrix Θ , the event of observing an edge between nodes i and j is independent of observing an edge between nodes i' and j' for $(i, j) \neq (i', j')$. Thus one can view the adjacency matrix A as a $n \times n$ array of independent Bernoulli random variables, with $\mathbb{P}(A_{ij} = 1 \mid \Theta) = P_{ij}$. Such an algorithm requires $O(n^2)$ effort.

However, the Kronecker structure in the edge probability matrix P can be exploited to sample a graph in expected $O(\log_2(n)|E|)$ time. The idea of Algorithm 1 suggested by Leskovec et al. (2010) is as follows:

The algorithm first determines the number of edges in the graph. Since the number of edges $|E| = \sum_{i,j} A_{ij}$ is the sum of n^2 independent Bernoulli random variables, it approximately follows the normal distribution for large n . Thus, one can sample the number of edges according to this normal distribution.

Next, the algorithm samples each individual edge according to the following recursive scheme. Let S denote the set of candidate nodes for the source and T the candidate target

nodes. Initially both S and T are $\{1, 2, \dots, n\}$. Using the matrix Θ , the proportion of expected number of edges in each quadrisection (north-west, north-east, south-west and south-east) of the adjacency matrix can be computed via

$$\sum_{i=an/2+1}^{(a+1)n/2} \sum_{j=bn/2+1}^{(b+1)n/2} P_{ij} \propto \theta_{ab}^{(1)}, \quad 0 \leq a, b \leq 1. \quad (5)$$

Then one can sample a pair of integers (a, b) , $0 \leq a, b \leq 1$, with the probability of (a, b) proportional to θ_{ab} , to reduce S and T to $\{an/2 + 1, an/2 + 2, \dots, (a + 1)n/2\}$ and $\{bn/2 + 1, bn/2 + 2, \dots, (b + 1)n/2\}$, respectively. Due to the Kronecker structure of the edge probability matrix P , repeating this quadrisection procedure d times reduces both S and T to single nodes $S = \{i\}$ and $T = \{j\}$, which are now connected by an edge. There is a small non-zero probability that the same edge is sampled multiple times. In this case the generated edge is rejected and a new edge is sampled (see pseudo-code in Algorithm 1).

Algorithm 1 Sampling Algorithm of Stochastic Kronecker Graphs

```

1: procedure KPGMSAMPLE( $\tilde{\Theta}$ )
2:    $E \leftarrow \emptyset$ 
3:    $m \leftarrow \prod_{k=1}^d (\theta_{00}^{(k)} + \theta_{01}^{(k)} + \theta_{10}^{(k)} + \theta_{11}^{(k)})$ 
4:    $v \leftarrow \prod_{k=1}^d ((\theta_{00}^{(k)})^2 + (\theta_{01}^{(k)})^2 + (\theta_{10}^{(k)})^2 + (\theta_{11}^{(k)})^2)$ 
5:   Generate  $X \sim \mathcal{N}(m, m - v)$ 
6:   for  $x = 1$  to  $X$  do
7:      $S_{start}, T_{start} \leftarrow 1$ 
8:      $S_{end}, T_{end} \leftarrow n$ 
9:     for  $k \leftarrow 1$  to  $d$  do
10:      Sample  $(a, b) \propto \theta_{ab}^{(k)}$ 
11:       $S_{start} \leftarrow S_{start} + a \left(\frac{n}{2^k}\right)$ 
12:       $T_{start} \leftarrow T_{start} + b \left(\frac{n}{2^k}\right)$ 
13:       $S_{end} \leftarrow S_{end} - (1 - a) \left(\frac{n}{2^k}\right)$ 
14:       $T_{end} \leftarrow T_{end} - (1 - b) \left(\frac{n}{2^k}\right)$ 
15:     end for
16:     # We have  $S_{start} = S_{end}, T_{start} = T_{end}$ 
17:      $E \leftarrow E \cup \{(S_{start}, T_{start})\}$ 
18:   end for
19:   return  $E$ 
20: end procedure

```

3 Multiplicative Attribute Graph Model

An alternate way to view KPGM is as follows: Associate the i -th node with a bit-vector $b(i)$ of length $\log_2(n)$ such that $b_k(i)$ is the k -th digit of integer $(i - 1)$ in its binary representation. Then one can verify that the (i, j) -th entry of the edge probability matrix P in (3) can be written as

$$P_{ij} = \prod_{k=1}^d \theta_{b_k(i) b_k(j)}^{(k)}. \quad (6)$$

Under this interpretation, one may consider $b_k(i) = 1$ (resp. $b_k(i) = 0$) as denoting the presence (resp. absence) of the k -th attribute in node i . The factor $\theta_{b_k(i) b_k(j)}^{(k)}$ denotes the probability of an edge between nodes i and j based on the value of their k -th attribute. The attributes are assumed independent, and therefore the overall probability of an edge between i and j is just the product of $\theta_{b_k(i) b_k(j)}^{(k)}$'s.

The Multiplicative Attribute Graph Model (MAGM) of Kim and Leskovec (2010) is also obtained by associating a bit-vector $f(i)$ with a node i . However, $f(i)$ need not be the binary representation of $(i - 1)$ as was the case in the KPGM. In fact, $f(i)$ need not even be of length $\log_2(n)$. We simply assume that $f_k(i)$ is a Bernoulli random variable with $\mathbb{P}(f_k(i) = 1) = \mu^{(k)}$. In addition to $\tilde{\Theta}$ defined in (4), the model now has additional parameters $\tilde{\mu} := \{\mu^{(1)}, \mu^{(2)}, \dots, \mu^{(d)}\}$, and the (i, j) -th entry of the edge probability matrix Q is written as

$$Q_{ij} = \prod_{k=1}^d \theta_{f_k(i) f_k(j)}^{(k)}. \quad (7)$$

4 Quilting Algorithm

A close examination of (6) and (7) reveals that KPGM and MAGM are very related. The only difference is that in the case of the KPGM the i -th node is mapped to the bit vector corresponding to $(i - 1)$ while in the case of MAGM it is mapped to an integer λ_i (not necessarily $(i - 1)$) whose bit vector representation is $f(i)$. We will call λ_i the *attribute configuration* of node i in the sequel.

For ease of theoretical analysis and in order to convey our main ideas we will initially assume that $d = \log_2(n)$, that is, we assume that $f(i)$ is of length $\log_2(n)$ or equivalently $0 \leq \lambda_i < n$ (this assumption will be relaxed in Section 4.2). Under the above assumption, every entry of Q has a corresponding counterpart in P because

$$Q_{ij} = P_{\lambda_i \lambda_j}. \quad (8)$$

The key difficulty in sampling graphs from MAGM arises because the attribute configuration associated with different nodes need not be unique. To sidestep this issue we partition the nodes into B sets such that no two nodes in a set share the same attribute configuration.

While a number of partitioning schemes can be used, we find the following scheme to be easy to analyze and efficient in practice: For each i define the set $Z_i := \{j \text{ s.t. } j \leq i \text{ and } \lambda_i = \lambda_j\}$. Clearly $|Z_i|$ counts nodes j whose index is smaller than or equal to i and which share the same attribute configuration λ_i . We now define the c -th set D_c in our partition as $D_c := \{i \text{ s.t. } |Z_i| = c\}$. No two nodes in D_c share the same attribute configuration. Furthermore, one can show that the number of sets B is minimized by this scheme.

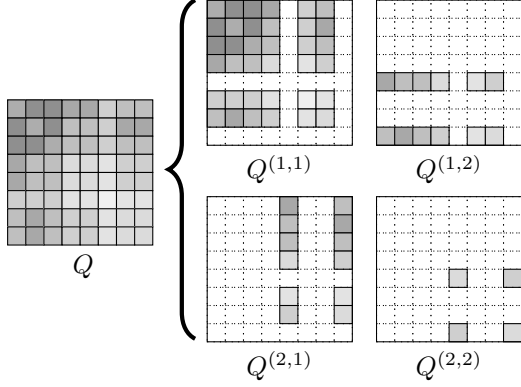


Figure 2: Partition the MAGM edge probability matrix Q into B^2 pieces such that no two nodes in a piece share the same attribute configuration.

Theorem 2 (Size Optimality of the Partition) *Let the partition of $\{1, \dots, n\}$ obtained by the above scheme be denoted as D_1, D_2, \dots, D_B . Then, B , the number of nonempty sets in the partition, is minimized.*

Proof See Appendix A. ■

Using the partition D_1, \dots, D_B , we can partition the edge probability matrix Q into B^2 sub-matrices (Figure 2):

$$Q_{i,j}^{(k,l)} = \begin{cases} Q_{i,j} & \text{if } i \in D_k, j \in D_l, \\ 0 & \text{otherwise.} \end{cases}$$

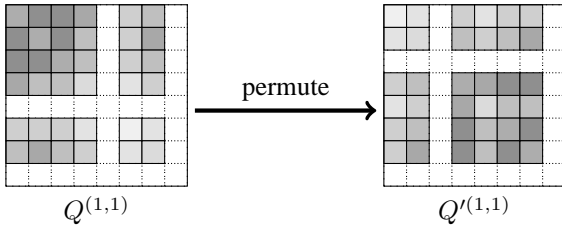


Figure 3: Each piece of the edge probability matrix is permuted to become a sub-matrix of the KPGM edge probability matrix. One can then apply Algorithm 1 to sample graphs from this permuted edge probability matrix and retain the sub-graph of interest.

Next, by applying a permutation which maps λ_i to i we can transform each of the B^2 sub-matrices of Q into a submatrix of P as illustrated in Figure 3. Formally, define

$$Q_{i,j}'^{(k,l)} = \begin{cases} Q_{x,y} & \text{if } x \in D_k, y \in D_l, i = \lambda_x, j = \lambda_y \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 1 can be used to sample graphs from this permuted edge probability matrix with parameters $\tilde{\Theta}$. We fil-

ter the sampled graph to only retain the sub-graph of interest. Finally, the sampled sub-graphs are un-permuted and quilted together to form a sample from the MAGM (see Figure 4). Let $A'^{(k,l)}$ denote the adjacency matrix of the graph sampled from the edge probability matrix $Q'^{(k,l)}$ via Algorithm 1. Define

$$A_{i,j}^{(k,l)} = \begin{cases} A'_{x,y} & \text{if } i \in D_k, j \in D_l, x = \lambda_i, y = \lambda_j \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

The quilted adjacency matrix A is given by $\sum_{k,l} A^{(k,l)}$. See Algorithm 2.

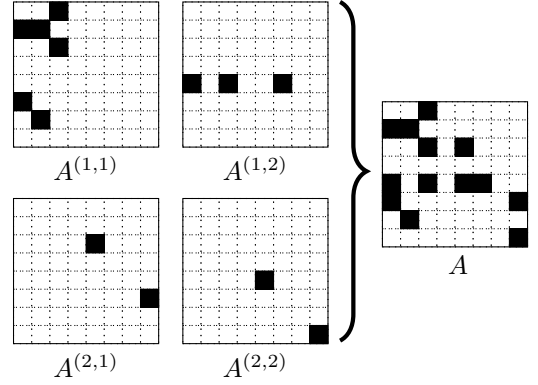


Figure 4: The sub-graphs sampled from the previous step are un-permuted and quilted together to form a graph sampled from the MAGM.

Algorithm 2 Sampling Algorithm of Multiplicative Attribute Graphs

```

1: function MAGSAMPLEEDGES( $\tilde{\Theta}, f(1), \dots, f(n)$ )
2:    $B \leftarrow \max_i |Z_i|$ 
3:   for  $k \leftarrow 1$  to  $B$  do
4:     for  $l \leftarrow 1$  to  $B$  do
5:        $E^{(k,l)} \leftarrow \text{KPGMSAMPLE}(\tilde{\Theta})$ 
6:       for each  $(x, y) \in E^{(k,l)}$  do
7:         if  $(i, j)$  such that  $i \in D_k, j \in D_l$ , and
            $x = \lambda_i, y = \lambda_j$  exists then
8:            $E \leftarrow E \cup \{(i, j)\}$ 
9:         end if
10:      end for
11:    end for
12:  end for
13:  return  $E$ 
14: end function
    
```

Theorem 3 (Correctness) *Algorithm 2 samples the entries of the adjacency matrix A independently with $\mathbb{P}(A_{ij} = 1 \mid \tilde{\Theta}, \lambda_1, \dots, \lambda_n) = Q_{ij}$.*

Proof See Appendix A. ■

4.1 Time Complexity

Since the expected running time of Algorithm 1 is $O(\log_2(n)|E|)$, the expected time complexity of quilting is clearly $O(B^2 \log_2(n)|E|)$. The key technical challenge in order to establish the efficiency of our scheme is to show that with high probability B is small, ideally $O(\log_2(n))$.

Balanced Attributes Suppose the distribution of each attribute is balanced, that is, $\mu^{(1)} = \mu^{(2)} = \dots = \mu^{(d)} = 0.5$ and $n = 2^d$. Define a random variable $X_c^i = 1$ if $\lambda_i = c$ and zero otherwise. Since $\mu^{(k)} = 0.5$, it follows that $\mathbb{P}(X_c^i = 1) = \frac{1}{2^d} = \frac{1}{n}$. If we let $Y_c = \sum_{i=1}^n X_c^i$, then clearly $B = \max_c Y_c$. Since X_c^i are independent Bernoulli random variables, Y_c is a binomial random variable which converges to a Poisson random variable with parameter 1 as $n \rightarrow \infty$. Using standard Chernoff bounds for the Poisson distribution (see Theorem 5), we can write

$$\mathbb{P}(Y_c > t) \leq \frac{e^t}{et^t}, \text{ and hence} \quad (10)$$

$$\mathbb{P}(B = \max_c Y_c > t) \leq \sum_{c=1}^n \mathbb{P}[Y_c > t] \leq \frac{ne^t}{et^t}. \quad (11)$$

Replacing t by $\log_2(n)$,

$$\mathbb{P}(B > \log_2(n)) \leq \frac{n^2}{e(\log_2(n))^{\log_2(n)}}. \quad (12)$$

As $n \rightarrow \infty$, (12) goes to 0 (also see Figure 5). Therefore we have

Theorem 4 When $\mu^{(1)} = \mu^{(2)} = \dots = \mu^{(d)} = 0.5$, and $n = 2^d$, with high probability the size of partitions B is smaller than $\log_2(n)$.

Unbalanced Attributes As before, we let $\mu^{(1)} = \mu^{(2)} = \dots = \mu^{(d)} = \mu$ and $n = 2^d$, but now we analyze the case when $\mu \neq 0.5$. By transposing some of $\Theta^{(k)}$ if necessary, it suffices to restrict our attention to $\mu \in (0.5, 1]$. We define the random variables X_c^i and Y_c as in the previous section. However, now $\mathbb{P}(X_c^i)$ depends on the number of 1's in the binary representation of c . In particular, if $c = 2^d = n$ then $\mathbb{P}(X_n^i) = \mu^{\log_2(n)}$ and $Y_n = n\mu^{\log_2(n)}$. Furthermore, $\mathbb{P}(X_c^i) < \mu^{\log_2(n)}$ for every $c \neq n$. Therefore, when μ is close to 1 and n is large, $B := \max_c Y_c$ equals $Y_n = n\mu^{\log_2(n)}$ with high probability (see Figure 6). The expected running time of our algorithm then becomes $(n^{\log_2(\mu)+1} \log_2(n)|E|)$.

4.2 Handling the Case When $n \neq 2^d$

To simplify the analysis assume $\mu^{(1)} = \dots = \mu^{(k)} = 0.5$.

$n > 2^d$ **Case** First, consider the case $n > 2^d$, and let $d' := \lceil \log_2(n) \rceil$, $d'' := \lfloor \log_2(n) \rfloor$. Each Y_c now

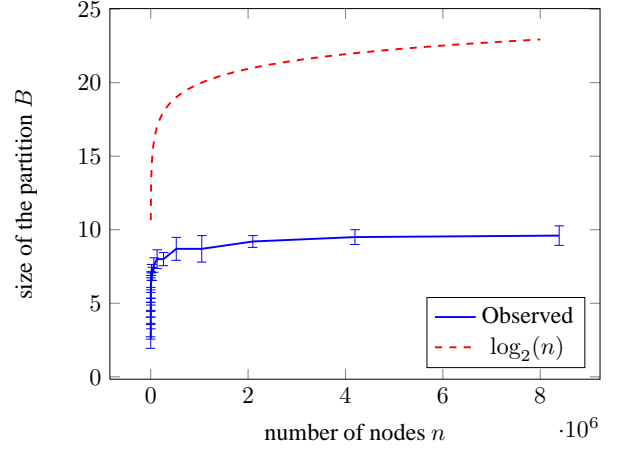


Figure 5: Number of nodes vs. size of the partition, when $\mu^{(k)}$'s are all set to be 0.5. For each n , we performed 10 trials and report average values (blue solid line). The red dashed line is the bound predicted by (12). Observe that in practice, the size of the partition grows much slower than $O(\log_2(n))$.

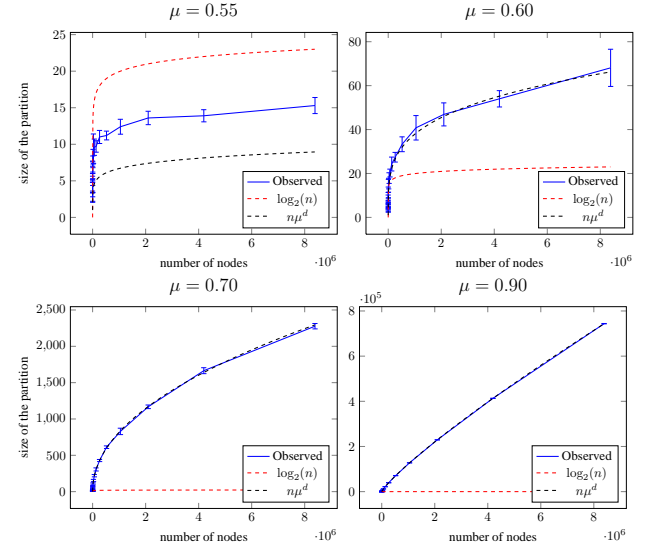


Figure 6: Number of nodes vs. size of the partition, when $\mu^{(k)}$'s are all set to be 0.55, 0.60, 0.70, and 0.90. Again, 10 number of F matrices were sampled for each n and μ , and the average size of the partition is taken. For small values of n , $n\mu^d$ approximation is not tight but the observed value is sandwiched between $\log_2(n)$ and $n\mu^d$. For $\mu > 0.70$, the $n\mu^d$ approximation is tight.

converges to the Poisson distribution with parameter $\frac{n}{2^d}$, and using the Chernoff bound again, one can prove that $B = O(2^{d'-d} \log_2(n))$ with high probability, for large n . For details, refer to Appendix C.

Therefore, the expected time complexity is bounded by $O((\log_2(n))^2 d |E|)$, and the algorithm gets faster as d decreases.

$n < 2^d$ Case For the sake of completeness we will make some remarks for the case when $n < 2^d$. However, Kim and Leskovec (2010) and Kim and Leskovec (2011) report that in practice $d \approx \log_2(n)$ usually results in the most realistic graphs. In general, for large d the value of B is small, but the number of edges in graphs sampled by Algorithm 1, which is called by line 5 of Algorithm 2, increases exponentially with d . Therefore, the overall complexity of the algorithm is at least $\Omega(4^{d-d''} \mathbb{E}[|E|])$, and thus naively applying Algorithm 2 is not advantageous when $d - d''$ is high. Our experiments in Section 6.4 confirm this behavior.

5 Speeding up the Algorithm

The key to speeding up our algorithm for the case when $\mu^{(k)} \neq 0.5$ is the following observation: When $\mu^{(k)}$ approaches 0 or 1, the number of distinct attribute configurations reduces significantly. For instance, when $\mu^{(k)}$ approaches 1 attribute configurations which contain more 1's in their binary representation are generated with greater frequency. Similarly, when $\mu^{(k)}$ approaches 0 the attribute configurations which contain more 0's in their binary representation are preferred. Figure 7 represents this phenomenon visually.

We select a number B' (see below) and collect all nodes i whose attribute configuration λ_i occurs at most B' times in the set $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ into a set W . Since each attribute configuration occurs at most B' times in W , the sub-graph corresponding to the nodes in W can be sampled in $O(B'^2 \log_2(n) |E|)$ by using Algorithm 2.

We partition the nodes whose attribute configuration occurs more than B' times into sets $\hat{D}_1, \dots, \hat{D}_R$ such that the attribute configuration of each node in \hat{D}_i is the same, say λ'_i . The sub-graph corresponding to each \hat{D}_i is an uniform random graph with probability of an edge being equal to $P_{\lambda'_i, \lambda'_i}$. On the other hand, the sub-graph corresponding to nodes \hat{D}_i and \hat{D}_j for $i \neq j$ is also an uniform random graph with the probability of an edge being equal to $P_{\lambda'_i, \lambda'_j}$. Finally, the sub-graph corresponding to a node i' in W and the set \hat{D}_j is an uniform random graph with the probability of an edge being equal to $P_{\lambda_{i'}, \lambda'_j}$. These sub-graphs can be sampled with $O((|W| + d)R + |E|)$, $O(dR + |E|)$, and $O(dR^2 + |E|)$ effort respectively¹.

¹ Instead of sampling k i.i.d. Bernoulli random variables

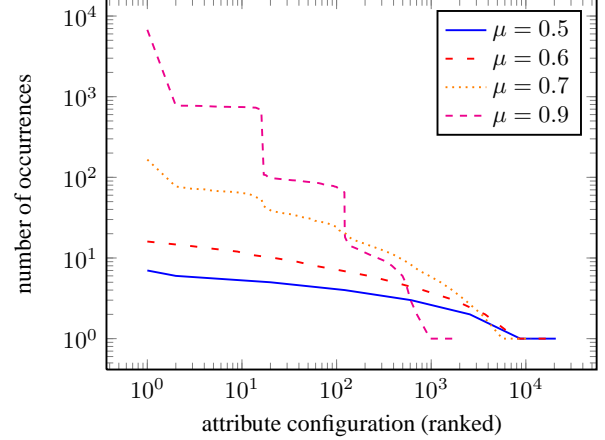


Figure 7: We rank attribute configurations based on their frequency of occurrence and plot them for different values of μ . We fixed $d = 15$, and $n = 2^{15}$ for this plot. When $\mu = 0.5$, the graph is very flat since every attribute configuration has the same probability $\frac{1}{2^d}$ of being sampled. On the other hand, when $\mu = 0.9$, the probability mass is very concentrated on a small number of configurations. Note that this is a log-log plot.

It remains to discuss how to choose the parameter B' . Towards this end let

$$T(B') = B'^2 \log(n) |E| + (|W| + d)R + dR^2.$$

Then, the overall time complexity of our algorithm is $O(T(B'))$. We calculate $T(B')$ for every B' , and choose the value which minimizes $T(B')$. Since there are only n distinct values of B' , this procedure requires $O(n)$ time.

6 Experiments

We empirically evaluated the efficiency and scalability of our sampling algorithm. Our experiments are designed to answer the following questions: Does our algorithm produce graphs with similar characteristics as observed by Kim and Leskovec (2010). How does our algorithm scale as a function of n , the number of nodes in the graph. Furthermore, since our theoretical analysis assumed $\mu = 0.5$ and $d = \log_2(n)$ we were interested in the following additional questions: How does the algorithm behave for $\mu \neq 0.5$. How does our algorithm scale when the number of features d is different from $\log_2(n)$.

Our code is implemented in C++ and will be made available for download from <http://www.stat.purdue.edu/~yun3>. All experiments are run on a machine with a 2.1 GHz processor running Linux. For the first three experiments we

X_1, X_2, \dots, X_k with parameter p , we use a geometric distribution with parameter p to sample to generate random variables K_j such that $1 \leq K_1 < K_2 < \dots \leq k$, and set $X_{K_j} = 1$.

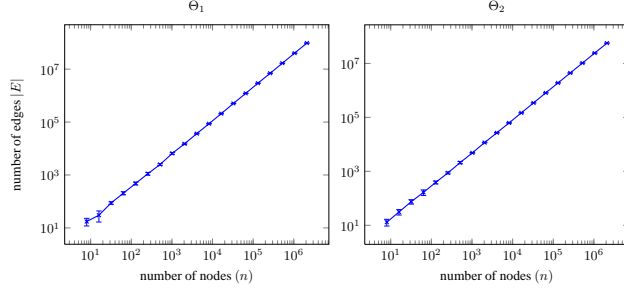


Figure 8: The number of edges $|E|$ as a function of the size n of the graphs sampled from the MAGM for two different values of Θ . The near linear rate of growth on the log-log plots confirms the observation that $|E| = n^c$ for some constant c .

uniformly set $n = 2^d$, where n is the number of nodes in the graph and d is the dimension of the features. We used the same Θ matrices at all levels, that is, we set $\Theta = \Theta^{(1)} = \Theta^{(2)} = \dots = \Theta^{(k)}$. Furthermore, we experimented with the following Θ matrices used by Kim and Leskovec (2010) and Moreno and Neville (2009) respectively:

$$\Theta_1 = \begin{bmatrix} 0.15 & 0.7 \\ 0.7 & 0.85 \end{bmatrix} \text{ and } \Theta_2 = \begin{bmatrix} 0.35 & 0.52 \\ 0.52 & 0.95 \end{bmatrix} \quad (13)$$

6.1 Properties of the Generated Graphs

Theorem A guarantees that our algorithm generates valid graphs from the MAGM. In our first experiment we also verify this claim empirically. Towards this end, we set $\mu = 0.5$ and generated graphs of size $n = 2^d$ for various values of d . For each n we repeated the sampling procedure 10 times and studied various properties of the generated graphs. As reported by Kim and Leskovec (2010), the number of edges $|E|$ in the graphs generated by MAGM grow as $|E| = n^c$ for some constant c . Graph samples generated by our algorithm also confirm to this observation, as can be seen in Figure 8. Furthermore, Kim and Leskovec (2010) report that the proportion of nodes in the largest strong component increases asymptotically to 1. We also observe this behavior in the samples generated by our algorithm (see Figure 9). These experiments indeed confirm that our algorithm samples valid graphs from the MAGM.

6.2 Scalability

To study the scalability of our algorithm we fixed $\mu = 0.5$ and generated 10 graphs of size $n = 2^d$ for various values of d . Figure 10 compares the running time of our algorithm vs a naive scheme which uses n^2 independent Bernoulli trials based on the entries of the edge probability matrix.

Note that using the naive sampling scheme we could not sample graphs with more than 262,144 nodes in less than 8

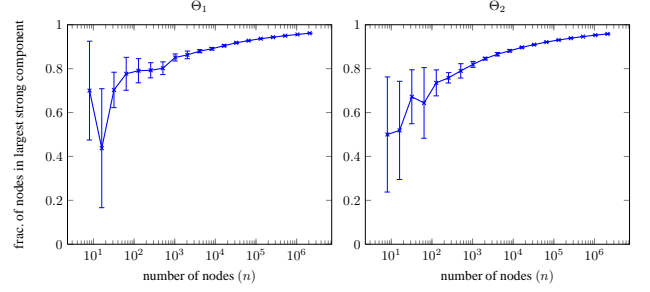


Figure 9: The fraction of nodes in the largest strong component as a function of the size n of the graphs sampled from the MAGM for two different values of Θ . Asymptotically, the fraction of edges approaches 1 implying that the entire graph is part of the same strong component.

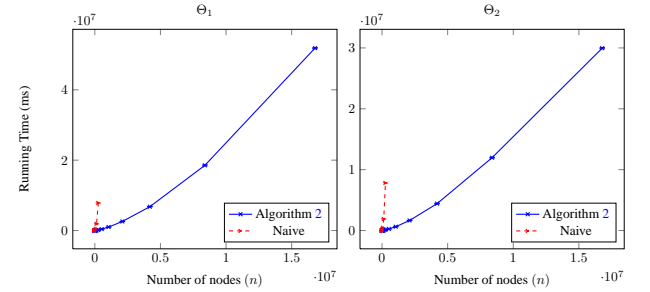


Figure 10: Comparison of running time (in milliseconds) of our algorithm vs the naive sampling scheme as a function of the size n of the graphs sampled from the MAGM for two different values of Θ .

hours. In contrast, the running time of our algorithm grows significantly slower than $O(n^2)$ and consequently we were able to comfortably sample graphs with a million nodes in less than twenty minutes. The largest graphs produced by our algorithm contain over 8 million nodes (8,388,608) and 20 billion edges. In fact these graphs are, to the best of our knowledge, at least 32 times larger than the largest MAGM graphs reported in literature in terms of number of nodes. Furthermore, we observed that our algorithm exhibits the same behavior across a range of Θ values (not reported here). To further demonstrate the scalability of our algorithm we plot the running time of our algorithm normalized by the number of edges in the graph in Figure 11. Across a range of n values, our algorithm spends a constant time for each edge that is generated. We can therefore conclude that the running time of our algorithm grows empirically as $O(|E|)$.

6.3 Effect of μ

Our theoretical analysis (Theorem 4) guarantees the scalability of our sampling algorithm for $\mu = 0.5$. In this section we explore empirically how the running time of our algo-

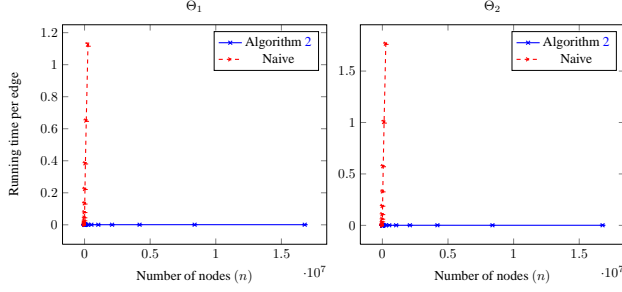


Figure 11: Running time per each edge (in milliseconds) for our algorithms vs the naive algorithm as a function of n the number of nodes. Note that the normalized running time of our algorithm is nearly constant and does not change as n increases.

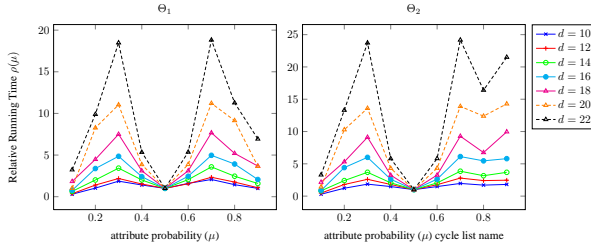


Figure 12: Relative running time $\rho(\mu)$ for two different values of Θ .

algorithm varies as we vary μ . Towards this end we define and study the relative running time $\rho(\mu) := \frac{T(\mu)}{T(0.5)}$, where $T(\mu)$ denotes the running time of the algorithm as a function of μ . In Figure 12 we plot $\rho(\mu)$ for different values of $n = 2^d$.

As expected, our algorithm performs well for $\mu = 0.5$, in which case the size of partition B is bounded by $\log_2(n)$ with high probability. Similarly, when $\mu \approx 0$ or 1 the attribute configurations have significantly less diversity and hence sampling becomes easy. However, there is also a tendency that the running time increases as μ increases. This is because the number of edges is also a function of μ , and due to our choice of Θ it is an increasing function. This phenomenon is more conspicuous for Θ_2 than Θ_1 , since θ_{11} of the former is larger than that of the latter.

One may also be interested in $\rho_{\max} := \max_{0 \leq \mu \leq 1} \rho(\mu)$. In order to estimate ρ_{\max} we let $\mu \in \{0.1, 0.2, \dots, 0.9\}$ and plotted the worst value of $\rho(\mu)$ as a function of n the number of nodes in Figure 13. In all cases ρ_{\max} was attained for $\mu = 0.7$ or $\mu = 0.9$. It is empirically seen that the factor $\rho(\mu)$ increases as the number of nodes n increases, but the speed of growth is reasonably slow such that still the sampling of graphs with millions of nodes is feasible for any value of μ .

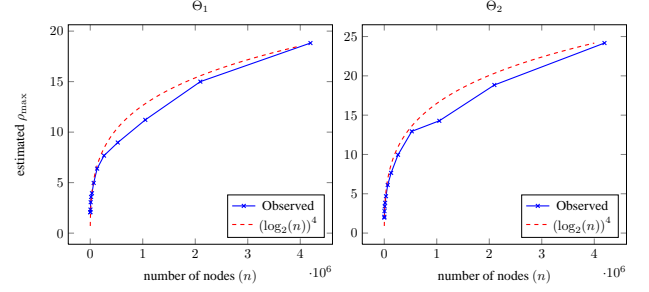


Figure 13: Estimated ρ_{\max} as a function of the number of nodes for two different values of Θ .

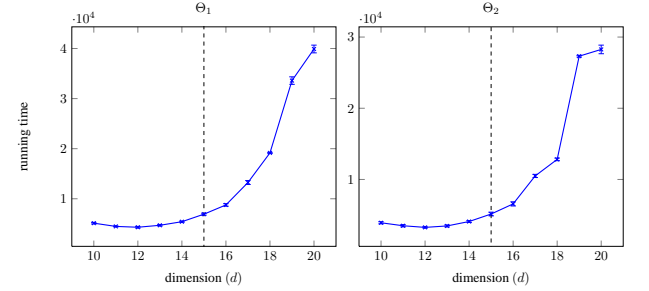


Figure 14: The effect of dimension d on the running time, where other parameters are fixed as $\mu = 0.5$ and $n = 2^{15}$. $d = 15$ (when $n = 2^d$) is highlighted by the dashed line.

6.4 Effect of d

Now we study how the performance of our model varies as d changes. In particular we fix $n = 2^{15}$ and vary d to investigate its effect on running time of our algorithm. Figure 14 shows that there is no significant difference in the running time for $d \leq \log_2(n)$. However, as we explained in Section 4.2, the running time of our algorithm increases exponentially when $d > \log_2(n)$.

7 Conclusion

We introduced the first sub-quadratic algorithm for efficiently sampling graphs from the MAGM. Under technical conditions, the expected running time of our algorithm is $O((\log_2(n))^3 |E|)$. Our algorithm is very scalable and is able to produce graphs with approximately 8 million nodes in under 6 hours. Even when the technical conditions of our analysis are not met, our algorithm scales favorably. We are currently working on rigorously proving the performance guarantees for the case when $\mu \neq 0.5$.

Efficiently sampling MAGM graphs for the case when $d \geq \log_2(n)$ remains open. We are currently investigating how high-dimensional similarity search techniques such as locality sensitive hashing (LSH) or inverse indexing can be applied to this problem.

References

- Dennis S. Bernstein. *Matrix Mathematics*. Princeton University Press, 2005.
- P.D. Hoff. Multiplicative latent factor models for description and prediction of social networks. *Computational & Mathematical Organization Theory*, 15(4):261–272, 2009. ISSN 1381-298X.
- D.R. Hunter, S.M. Goodreau, and M.S. Handcock. Goodness of fit of social network models. *Journal of the American Statistical Association*, 103(481):248–258, 2008.
- Myunghwan Kim and Jure Leskovec. Multiplicative attribute graph model of real-world networks. *CoRR*, abs/1009.3499, 2010.
- Myunghwan Kim and Jure Leskovec. Modeling social networks with node attributes using the multiplicative attribute graph. In *UAI*, 2011.
- Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11(Feb):985–1042, 2010.
- S. Moreno and J. Neville. An investigation of the distributional characteristics of generative graph models. In *Proceedings of the The 1st Workshop on Information in Networks*. Citeseer, 2009.
- G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social Networks*, 29(2):173–191, 2007. ISSN 0378-8733.
- S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nat Genet*, 31(1):64–68, May 2002.

A Technical Proofs

Proof of Theorem 2 Let $i' := \operatorname{argmax}_i |Z_i|$. The attribute configuration $\lambda_{i'}$ corresponding to node i' appears at least B times in the set $\{\lambda_1, \dots, \lambda_n\}$. By the pigeon-hole principle any partition of the set $\{1, \dots, n\}$ which contains less than B sets must have a set which contains two nodes i and j with attribute configuration $\lambda_i = \lambda_j = \lambda_{i'}$. Therefore, the number of sets in the partition must be at least B . Our partitioning scheme produces exactly B sets, and is therefore optimal.

Proof of Theorem By definition,

$$A_{i,j} = \sum_{1 \leq k, l \leq B} A_{i,j}^{(k,l)}. \quad (14)$$

To prove the theorem, we first show that $A_{i,j} = A_{\lambda_i, \lambda_j}^{|Z_i|, |Z_j|}$. This is straightforward from definition (9), since D_1, \dots, D_B is a partition of nodes and thus $i \in D_k$, $j \in D_l$ for only $k = |Z_i|$ and $l = |Z_j|$. This also implies

$$\begin{aligned} \mathbb{P}(A_{ij} = 1 \mid \tilde{\Theta}, \lambda_1, \dots, \lambda_n) &= A_{\lambda_i, \lambda_j}^{|Z_i|, |Z_j|} \\ &= P_{\lambda_i, \lambda_j} = Q_{i,j}, \end{aligned}$$

using (8).

To prove independence, we show that if $(i, j) \neq (i', j')$, then $(\lambda_i, \lambda_j) \neq (\lambda_{i'}, \lambda_{j'})$ or $(|Z_i|, |Z_j|) \neq (|Z_{i'}|, |Z_{j'}|)$. Since we already showed $A_{i,j} = A_{\lambda_i, \lambda_j}^{|Z_i|, |Z_j|}$, this implies independence of $A_{i,j}$ to other entries in A .

Now, suppose $(\lambda_i, \lambda_j) = (\lambda_{i'}, \lambda_{j'})$, since if it does not hold there is nothing to prove. Because $(i, j) \neq (i', j')$ by assumption, at least one of $i \neq i'$ or $j \neq j'$ is true. Without loss of generality, suppose $i \neq i'$. Then, since $\lambda_i = \lambda_{i'}$, $|Z_i| \neq |Z_{i'}|$ from definition of Z_i .

B Chernoff Bound of Poisson Distribution

Theorem 5 Let X be the random variable which is distributed as Poisson distribution of parameter λ . Then,

$$P(X \geq x) \leq \frac{e^{-\lambda} (e\lambda)^x}{x^x}. \quad (15)$$

C Upper bound of size of the partition when $n > 2^d$

As a binomial distribution with finite mean in limit, Y_c is approximately distributed as a Poisson distribution of parameter $\frac{n}{2^d}$. To use Chernoff bound (15) with $\lambda = \frac{n}{2^d}$, $x = 2^{t+1} \log_2(n)$, $t = d'' - d$, we first bound each term:

$$e^{-\lambda} = e^{-\frac{n}{2^d}} \leq e^{-2^t}, \quad (16)$$

$$e^x = e^{2^{t+1} \log_2(n)} = n^{2^{t+1}}, \quad (17)$$

$$\lambda^x = \left(\frac{n}{2^d}\right)^x \leq (2^{t+1})^{2^{t+1} \log_2(n)}, \quad (18)$$

$$x^x = (2^{t+1} \log_2(n))^{2^{t+1} \log_2(n)}. \quad (19)$$

Then, plugging these into (15),

$$\mathbb{P}(B = \max Y_c > 2^{t+1} \log_2(n)) \leq \sum_{c=1}^n \mathbb{P}(Y_c > 2^{t+1} \log_2(n)) \quad (20)$$

$$\leq n \cdot \frac{e^{-2^t} \cdot n^{2^{t+1}} \cdot (2^{t+1})^{2^{t+1} \log_2(n)}}{(2^{t+1} \log_2(n))^{2^{t+1} \log_2(n)}}. \quad (21)$$

By taking log,

$$\begin{aligned} \log \mathbb{P}(B = \max Y_c > 2^{t+1} \log_2(n)) &\leq -2^t + 2^{t+1} \log_2(n) \\ &\quad + (2^{t+1} \log_2(n)) \log_2 2^{t+1} - 2^{t+1} \log_2(n) \log_2(2^{t+1} \log_2(n)) \\ &= -2^t + 2^{t+1} \log_2(n) (1 + \log_2 2^{t+1} - \log_2(2^{t+1} \log_2(n))), \end{aligned} \quad (22)$$

and this goes to $-\infty$ as $n \rightarrow \infty$. Therefore, $\mathbb{P}[B = \max Y_c > 2^{t+1} \log_2(n)] \rightarrow 0$.